Transient and Steady-State Statistical Analysis for Discrete Event Simulators

Transient and Steady-State Statistical Analysis for Discrete Event Simulators

Stephen Gilmore[1], Daniël Reijsbergen[1], and Andrea Vandin[2]

[1] University of Edinburgh, Edinburgh, Scotland
[2] IMT School for Advanced Studies, Lucca, Italy

Abstract. We extend the model checking tool MultiVeStA with statistical model checking of steady-state properties. Since MultiVeStA acts as a front-end for simulation tools, it confers this ability onto any tool with which it is integrated. The underlying simulation models are treated as black-box systems. We will use an approach based on *batch means* using the ASAP3 algorithm. We motivate the work using two case studies: a biochemical model written in the Bio-PEPA language and an application from transport logistics.

Keywords: statistical model checking, steady-state, batch means, MultiVeStA

1 Introduction

Statistical model-checking (SMC) [26,38] is a verification technique which is used for checking logical properties of formal models of large-scale systems. Based on analysis by simulation, statistical model-checking offers advantages over explicit-state model-checking because the absence of a formal representation of the reachable state-space of the system means that the approach scales better in supporting more detailed models of more complex systems, albeit at the cost of qualifying results with a statistical confidence. The SMC approach is well-suited to investigating transient (meaning, time-dependent) properties of systems.

Our focus is on *black-box systems*, which are those for which no knowledge of the internal state or transition structure is assumed. For these systems, conclusions about the satisfaction of formal system properties can only be based on observation traces, which in our case are obtained using system simulation. In this paper we extend an existing statistical model checker for black-box systems, MultiVeStA [31], by adding the ability to check steady-state (meaning, time-independent) properties of systems. The application of steady-state model checking to black-box systems is novel and is the main contribution of this paper.

Previous applications of SMC for black-box systems [32,37,11] have considered only transient properties. Conversely, existing SMC procedures for steady-state properties are only applicable to Markov chain models and the associated logics such as CSL [7] and UTSL [37], and are hence not sufficiently generic for black-box systems.

In particular, the current leading SMC model-checking techniques for steady-state analysis are the approach based on *regeneration cycles* which is implemented in the MRMC model-checker [23], and the approach based on *perfect simulation* which is

found in [18]. These are not applicable in the context of black-box systems for the following reasons.

- The use of regeneration cycles requires that the state space has a pre-identified bottom strongly connected component (BSCC) structure – i.e., state space regions for which the probability of travelling from one region to another is 0 – and assumes that the system has a known regeneration state, which is not true for black-box systems, thereby making this approach inapplicable for our setting.
- The perfect simulation algorithm of [18] will exhibit either a time complexity which is linear in the state space size, or requires envelope computation which means that it is prohibitively expensive for use with black-box systems [12].

In this paper, we use the method of *batch means* [17,3]. Typical challenges facing the batch means method include the warm-up period, independence of the batches, and getting 'stuck' in a BSCC when more than one exists. To ameliorate these, we use an approach based on the ASAP3 [34] algorithm, which prescribes that we continue sampling until the batch means pass two statistical tests, one for normality and one for the absence of correlation between the means. To avoid the (rare) multiple-BSCC problem, one could further extend the algorithm with the method of independent replications [2].

The MultiVeStA tool which we extend here is a parametric simulation analyser which functions as a front-end for multiple discrete-event simulators by interactively asking them to produce observation traces which are then analysed. The logical query language underlying MultiVeStA, called MultiQuaTEx, allows the specification of expressions which map functions of state variables onto real numbers, thereby defining stochastic processes. In combination with a stopping criterion for individual runs, this defines a property specification language that generalises (the transient fragments of) other languages such as the logics PCTL [22] and CSL [6,7] for Markov chains, and UTSL [37] for general discrete-event systems (see discussion in [1]). However, MultiQuaTEx and UTSL currently both omit *steady-state* properties, which do not involve the value of a random variable at a (possibly random) stopping time, but rather its long-run average.

The parametric multi-simulator approach of MultiVeStA is validated using two case studies. The first is a model of local intracellular signalling reactions in the cAMP/PKA/MAPK pathway in neurons as studied in [27,15]. This model is introduced in Section 4.1 to facilitate explanation of the core concepts of MultiVeStA in the following sections. The second case study concerns a model for the performance analysis of a public transportation network in Edinburgh [29], parameterised using real-world GPS data. It is introduced in Section 5.

Information on how to replicate the experiments of this paper are available online at `http://sysma.imtlucca.it/tools/multivesta/batchMeans`.

2 Batch Means Method

The type of property that we are interested in is the steady-state value of a random variable F. To evaluate this property, we ask the simulator to generate a simulation run

Algorithm 1 Batch means algorithm

Require: B even (default: 256), variable of interest x
1: $T \leftarrow$ `determineInitialBatchDuration()`
2: $\mu \leftarrow (\mu_1, \ldots, \mu_B)$
3: **for** $i \in \{1, \ldots, B\}$ **do**
4: $\quad \mu_i \leftarrow$ `drawBatch`(x, T)
5: **end for**
6: $(a, \rho, d) \leftarrow$ `performGoodnessOfFitTests`(μ)
7: **while** $a > a^*$ `and` $\rho > \rho^*$ `and` $d > \delta$ **do**
8: $\quad$ **for** $i \in \{1, \ldots, B/2\}$ **do**
9: $\quad\quad \mu_i \leftarrow (\mu_{2i} + \mu_{2i+1})/2$
10: $\quad$ **end for**
11: $\quad T \leftarrow 2T$
12: $\quad$ **for** $i \in \{B/2 + 1, \ldots, B\}$ **do**
13: $\quad\quad \mu_i \leftarrow$ `drawBatch`(x, T)
14: $\quad$ **end for**
15: $\quad (a, \rho, d) \leftarrow$ `performGoodnessOfFitTests`(μ)
16: **end while**
17: **return** confidence interval based on μ, compensating for correlation

$(F(t))_{t \geq 0}$ such that $F(t)$ represents the value of F when the simulated time is t. Then we define the steady-state mean π_F as

$$\pi_F = \lim_{T \to \infty} \frac{1}{T} \int_0^T F(t) dt.$$

Using batch means, π_F is estimated for a large value of T rather than for the limit. Let B and b be positive integers (two parameters). A confidence interval for this value is then constructed by dividing the time interval $[0, T]$ into B batches of equal time length. We discard the first b batches to remove initialisation bias, and assume that the means of the remaining batches are normally distributed. The challenge is to find a T large enough (but minimal) for the assumptions of normality and lack of initialisation bias to be approximately valid.

We solve this using an approach based on the ASAP3 procedure [34]: we draw a moderately large number of events (4096 by default) and record the obtained simulated time T_1. We then draw B (default value: 256) batches of time length T_1 and discard the first b (default value: 4) batches. We then perform a test for normality on the remaining batches — we use the Anderson-Darling test, offered by the SSJ library [24,25]. If the test fails to reject the null hypothesis that the batch means are drawn from the normal distribution, we compute the correlation between subsequent batch means and determine whether this is smaller than a threshold value given in [34]. If so, we construct a confidence interval that corrects for this correlation. If not, we iteratively repeat the experiment for $T_i = 2T_{i-1}$ until both conditions are met. When both conditions are met, we continue sampling (increasing the time length of the batches) until the confidence interval width is smaller than δ, yielding a Chow-Robbins confidence interval that is asymptotically valid [21].

A pseudocode representation of such algorithm appears in Algorithm 1. In particular, Algorithm 2 details how the size of batches is initially computed, i.e., how the parameter T_1 mentioned above is calculated. Instead, Algorithm 3 details how the statistical quality of the batches is computed.

Algorithm 2 `determineInitialBatchDuration()`

Require: initial event number X (default: 4096)
$T \leftarrow 0$
$k \leftarrow X$
while $T = 0$ **do**
  for $i \in \{1, \ldots, k\}$ **do**
    `simulator.takeStep()`
  end for
  $T =$ `simulator.getTime()`
  $k \leftarrow 2k$
end while
return T

Algorithm 3 `performGoodnessOfFitTests()`

Require: batch means μ,
$\mu' \leftarrow (\mu_{b+1}, \ldots, \mu_B)$
$a \leftarrow$ `PerformNormalityTest`(μ')
$\rho \leftarrow$ `PerformTestForAutocorrelation`(μ')
$d \leftarrow$ `ComputeConfidenceIntervalWidth`(μ')
return (a, ρ, d);

If the limit does not exist, then the confidence interval is expected to asymptotically widen, meaning that the procedure will not terminate. If the limit is not unique in the sense that different runs of the same procedure will yield different limits, owing to, for example, the existence of more than a single BSCC, then the method of *independent replications* [2] is to be preferred. We would like to stress again that the multiple-BSCC setting is rare in practice, and that the batch means is much faster than independent replications because the b warm-up batches only need to be drawn once rather than for every sample. In a general approach, the two methods can be integrated by starting with the method of independent replications, and switching to the method of batch means if the samples are judged to be sufficiently similar by a normality test.

Although our implementation allows experienced users to override the default parameters, it is not necessary for novice users to do so. There is typically no need to adjust the parameters B, b, and the initial number of events, because these only determine the number of initial runs; if this is chosen too low, the algorithm is designed to draw more samples. For the confidence level of the normality test, it is sufficient to choose a default value of 95% or 99%. (The confidence level for this test does not

provide a strict bound anyway because it is conducted several times. In the early stages of the simulation, when the means are typically very unlike the normal distribution and strongly correlated, the test's p-value and therefore the probability of stopping prematurely tends to be far below the threshold.) The only parameters that in all cases need to be set by the user are the confidence interval final width δ and confidence level α, which need to be set for most SMC procedures. Note that it is important to avoid choosing δ too low, because making it y times smaller will require roughly y^2 times as many samples.

3 MultiVeStA

This section introduces MultiVeStA [31], a Java framework for statistical model checking that can be easily integrated with existing discrete event simulators. MultiVeStA has been successfully applied to many scenarios, including: (by external users) contract-oriented middlewares [8], opportunistic network protocols [5], online planning [10], (by MultiVeStA's developers) software product lines [35,36], crowd-steering [28], public transportation systems [20,13], volunteer clouds [30], and swarm robotics [9].

MultiVeStA has a distributed architecture, making it possible for users to distribute simulations across a network of compute-servers. An in-depth discussion of MultiVeStA's architecture and of MultiQuaTEx is provided in [31,28].

The tool extends VeStA [1,33] and PVeStA [4], as discussed in [31]. More information on MultiVeStA and on the currently integrated simulators is available at `http://sysma.imtlucca.it/tools/multivesta/`.

3.1 Simulator integration

MultiVeStA adopts a distinctive black-box approach: it does not require systems to be specified in a given system specification language, but rather it makes it possible to directly analyse models written for simulators which have been integrated with MultiVeStA. In particular, MultiVeStA interacts with underlying simulators by triggering basic actions such as:

- `reset`: reset the simulator to its "initial state", and update the seed used for pseudo-random sampling, necessary before performing a (new) simulation;
- `next`: perform one step of simulation; and
- `eval`: evaluate an observation in the current simulation state.

This is obtained by instantiating MultiVeStA's Java interface which contains three corresponding methods. As a consequence, MultiVeStA natively supports Java-based simulators. However, it has been also integrated with C-based simulators using the Java Native Interface (JNI) or Python-based simulators using the `py4j` libraries.

Similar in spirit to MultiVeStA is (the independently proposed) Plasma-lab [11], a framework for SMC. The main difference lies in the property specification languages which are used (see [31]): temporal logics to estimate probabilities by Plasma-Lab, and

```
1  public class SimulatorState extends NewState{
2
3          //Reference to the simulator
4          private Simulator simulator;
5
6          /* @param params Parameters provided to MultiVeStA */
7          public SimulatorState(ParametersForState params) {
8                  super(params);
9                  //Name of the (file with the) model to be analyzed
10                 params.getModel();
11                 //Optional string with simulator-specific parameters.
12                 params.getOtherParameters();
13                 //Optional evaluator of model-specific observations see [31]
14                 params.getStateEvaluator();
15                 //Read the model and initialize the simulator...
16         }
17
18         /* @param randomSeed The new seed for random generation */
19         public void setSimulatorForNewSimulation(int randomSeed) {
20                 simulator.reset(randomSeed);
21         }
22
23         public void performOneStepOfSimulation() {
24                 simulator.performOneStep();
25         }
26
27         /* @param obs a string representing the observation to be evaluated
28          * @return the value of the evaluated observation */
29         public double rval(String obs) {
30                 simulator.eval(obs);
31         }
32 }
```

Listing 1. A skeleton of the adaptor between MultiVeStA and a simulator.

MultiQuaTEx by MultiVeStA (Section 4.2). Plasma-lab offers further statistical analysis techniques including sequential hypothesis testing for transient properties, but crucially it does not support steady-state model checking, which is the main contribution of this paper.

After adding the MultiVeStA library to the classpath of the simulator, one has to extend the class NewState in order to provide the three functionalities above. Listing 1 provides a skeleton of such class. Essentially, this is an adaptor between MultiVeStA and the simulator, which just propagates the requests received from MultiVeStA to the simulator, and returns the obtained results. We note that the class stores a reference to the simulator (Line 4), in the form of an object of a class in the simulator's namespace.

The class constructor (Lines 7-16) should perform only actions that have to be done once (and not for individual simulation runs). For example, it might read the model from a file. The parameters provided by the user to MultiVeStA are collected in the object `params`, containing the name of the model to be analysed, simulator-specific parameters, and a model-specific state evaluator (see [31] for more details on the latter).

The method `setSimulatorForNewSimulation` (Lines 19-21) is invoked by MultiVeStA to reset the simulator for a new simulation, providing also the new random seed. During a simulation, the method `performOneStepOfSimulation` (Lines 23-25) is invoked by MultiVeStA to perform a simulation step.

The method `rval` (Lines 29-31) is invoked by MultiVeStA to evaluate observations of the current state of a simulation run. Typically, the method delegates the evaluation of

the observations to the simulator, if possible, or deals only with observations common to all possible models that can be defined for the simulator (e.g., the current simulation time or the molecule count of a given chemical species in biological simulators), and invokes the state evaluator which is provided for any model-specific observations.

The state evaluator is not necessary if we know in advance all observations that can be computed for any model definable for the simulator, or if the evaluation of an observation can be delegated to the simulator. We have found in practice that the delegation of the evaluation to the simulator is often possible, and is the preferred option.

4 Analysis of a biochemical pathway using MultiVeStA

In this section we show how transient and steady-state analysis of a well-known biological model from the literature can be performed using MultiVeStA.

4.1 Example 1: the cAMP/PKA/MAPK biochemical pathway

As our first example in this paper we consider a biochemical reaction pathway expressed in Bio-PEPA [16], a process-algebra-based framework for the modelling and the analysis (including stochastic simulation) of biochemical networks. The Bio-PEPA software [14] supports a range of both continuous and discrete simulators including accelerated stochastic simulation methods. MultiVeStA interoperates with the Direct Method implementation of the Gillespie stochastic simulation algorithm provided by the Bio-PEPA Eclipse Plugin as discussed in [20]. The particular Bio-PEPA model considered is the cAMP/PKA/MAPK pathway modelled as in [27], analysed with Bio-PEPA in [15]. A schematic presentation of the pathway is given in Figure 1, taken from [15].

Three sub-networks can be identified in the pathway, highlighted in boxes with different colours. Each sub-network regards the activation of three molecules, namely *AC* (*adenylate cyclase*), *PKA* (*protein kinase A*) and *MAPK* (*mitogen-activated protein kinase*). We refer to [15] for a detailed presentation of the model. Here, we will show how MultiVeStA can be used to study the dynamics in terms of *activation* and *deactivation* of the enzyme *AC*, both as a function of time, and at steady state.

4.2 Transient Analysis with MultiVeStA

MultiVeStA offers a powerful and flexible property specification language, MultiQuaTEx [31], which allows the modeller to express transient properties to be verified. Intuitively, a MultiQuaTEx query specifies a random variable (representing, e.g., the number of active *AC* molecules at a certain point in time during a simulation).

The expected value of a MultiQuaTEx query is estimated as the mean $\overline{x}$ of n samples (taken from n simulations), with n large enough (but minimal) to guarantee that we can construct a $(1-\alpha)\cdot 100\%$ confidence interval around $\overline{x}$ of width at most δ, for given α and δ. This means that, with a probability of at least $1-\alpha$, the actual expected value belongs to the random interval $[\overline{x}-\frac{\delta}{2}, \overline{x}+\frac{\delta}{2}]$.

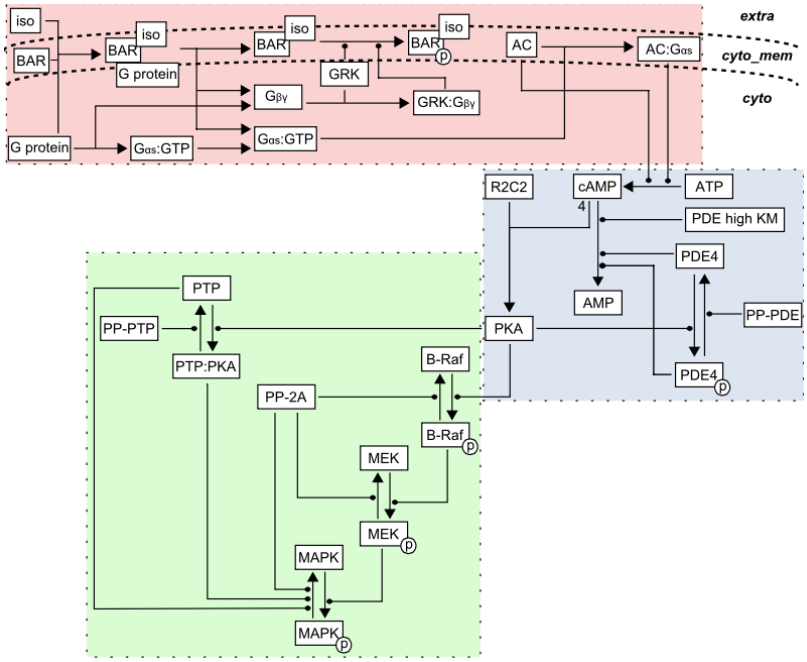

Fig. 1. Schematic representation of the cAMP/PKA/MAPK pathway (from [15]).

A MultiQuaTEx query might actually specify many random variables. An ensemble of simulation results computed by MultiVeStA can be reused multiple times to study each of the random variables in turn. Listing 2 depicts a MultiQuaTEx query for the cAMP/PKA/MAPK pathway.

```
1 fractionAtTime(t, act, inact) =
2         if{s.rval("time") >= t}
3             then s.rval(act) / (s.rval(act) + s.rval(inact))
4             else #fractionAtTime(t, act, inact)
5         fi;
6
7 eval parametric(E[fractionAtTime(t, "AC_active", "AC_inactive")],1.0,10.0,700.0);
```

Listing 2. A transient MultiQuaTEx query

We now overview MultiQuaTEx using the expression in Listing 2, defined for the above mentioned cAMP/PKA/MAPK pathway model. It studies the evolution over time of the fraction of active instances of adenylate cyclase (`AC_active`) over the total quantity of adenylate cyclase (`AC_active` + `AC_inactive`). A MultiQuaTEx query consists of a list of *MultiQuaTEx operators*, used in an `eval parametric` clause to specify the properties to be estimated.

Lines 1-5 define one MultiQuaTEx operator, `fractionAtTime`, with three parameters: `t`, `act` and `inact`. It is evaluated (in every simulation) as the fraction $\texttt{act}/(\texttt{act} + \texttt{inact})$ at time point `t`.

Line 7 instantiates `fractionAtTime`, specifying the properties to be evaluated: the (expected value of the) fraction of active *AC* from time point 1 to time point 700, with step 10. Many MultiQuaTEx operators can appear in an `eval parametric` clause, which is just syntactic sugar expanded into a list of `eval E[·]`. There will be 70 of these in the case of Listing 2, all evaluated using the same simulations.

A MultiQuaTEx operator consists of the following ingredients:

1. real-valued observations on the current simulation state (`s.rval`);
2. arithmetic expressions (Line 3);
3. conditional statements;
4. a one-step next operator which triggers the execution of a simulation step (the # symbol of Line 4);
5. recursion, used in Line 4 to evaluate the operator in the next simulation step.

This is general enough to express PCTL and CSL properties, as discussed in [1], however users must restrict themselves to queries that can be evaluated for each simulation in a finite number of simulation steps.

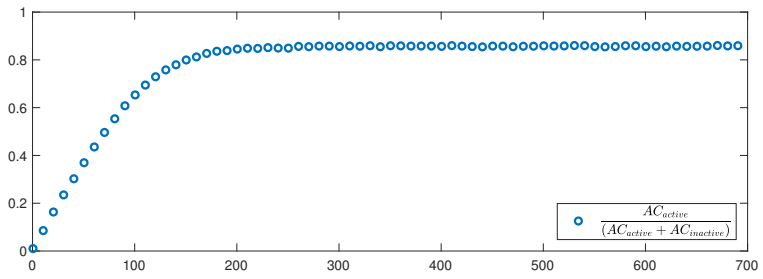

Fig. 2. Estimation of the transient MultiQuaTEx query from Listing 2.

Figure 2 depicts the estimation computed by MultiVeStA of the transient MultiQuaTEx query from Listing 2 for $\alpha = 0.05$ and $\delta = 0.01$ (meaning a 95% confidence interval of width at most 0.01). The analysis required 480 simulations, with a runtime of about 1000 seconds (without distributing the simulations) on a 2.6 GHz Intel Core i5 machine with 4 GB of RAM. A similar analysis has been performed in [15], studying how the fraction of active instances of *AC* (and of two additional molecules *PKA* and *MAPK*) changes over time. This has been done by running an arbitrary number of simulations with time horizon set to 700 seconds, and required a modification of the Bio-PEPA model to represent the fraction of active molecules directly within the model. Here, instead, we follow a separation-of-concerns approach, leaving the model unchanged and delegating to MultiQuaTEx the definition of the measures of interest. In addition, MultiVeStA gives a statistical assurance on the obtained measures.

4.3 Steady-State Analysis in MultiVeStA

The extension of MultiVeStA with steady-state capabilities required us to extend MultiQuaTEx correspondingly. Listing 3 provides a *steady-state* MultiQuaTEx query. It is similar to that in Listing 2, but it studies the fraction of active *AC* at steady state rather than at given time points.

```
1 fractionAtTime(act,inact) =
2         s.rval(act)/(s.rval(act)+s.rval(inact));
3
4 eval batchMeans(E[ fractionAtTime("AC_active","AC_inactive") ]) ;
```

Listing 3. A steady-state MultiQuaTEx query

The query is composed of two parts:

1. A list of MultiQuaTEx operators, where, differently from the transient case, the one-step next operator (#) is not allowed; and
2. the `eval batchMeans` clause, similar to the `eval parametric` one, which lists the properties to be estimated at steady state.

Intuitively, a steady-state MultiQuaTEx query defines a number of state observations. In order to estimate the value of each observation at steady state, we integrated within MultiVeStA the batch means methods described in Section 2. Intuitively, as depicted in Algorithm1, we do not perform n independent simulations as in the transient case, but only a single long simulation, divided in n batches of equal time length, each of which gives a sample.

The time horizon is automatically chosen to be large enough (but minimal), to guarantee that we can construct an $(1-\alpha)\cdot 100\%$ confidence interval $[\overline{x}-\frac{\delta}{2},\overline{x}+\frac{\delta}{2}]$ for the value at steady state of the property of interest, where $\overline{x}$ is the mean of the batch samples.

Figure 3 depicts the novel architecture of MultiVeStA. We see that MultiVeStA consists of two main macro-functionalities, transient analysis and steady-state one. Each functionality allows to parse and analyse the corresponding family of MultiQuaTEx queries. Both analysis interact with integrated simulators by using the adaptor interface discussed in Section 3.1. For easiness of presentation, the figure ignores the distributed architecture supporting the transient analysis.

Figure 2 shows that the fraction of active adenylate cyclase (seems to) stabilize to about 0.855 after 300 seconds of simulated time. We can now confirm that this actually holds at steady state. Indeed, Listing 3 is estimated as 0.855 using the same confidence interval used for the transient case ($\alpha = 0.05$ and $\delta = 0.01$). This has been done by performing a simulation with time horizon of 14066 seconds. Notably, the steady-state analysis took only about 20 seconds, compared to the 1000 seconds required by the transient one. The transient analysis is slower even in the case where we attempt to reduce the computation time of the simulation by setting 300 as the simulation stop-time in Listing 2, obtaining a runtime analysis of about 486 seconds.

Finally, we remark that when using the Bio-PEPA tools alone it is not possible to perform any form of steady-state analysis, thus our integration of MultiVeStA with

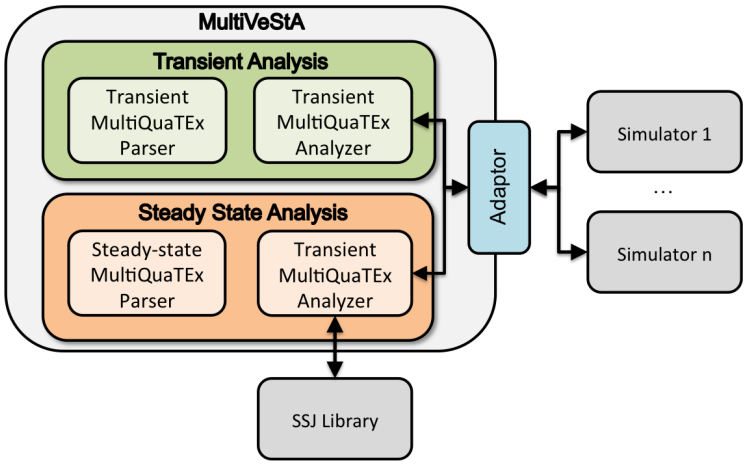

Fig. 3. The novel architecture of MultiVeStA

support for batch means provides analysis capabilities which are not provided by the Bio-PEPA Eclipse Plugin.

5 Example 2: Edinburgh Bus Simulator

Recently, many approaches have been provided to analyse models of public transportation systems parameterised using real-world GPS data (e.g., [19]). In this section we use MultiVeStA to study a model of a public city bus service in Edinburgh [29], obtained from GPS data. We consider the problem of guaranteeing a quality-of-service constraint required by the legislator. The recently submitted paper [29] presents a model of a bus service in Edinburgh, parameterised using real-world data. The model is used to evaluate the performance of bus networks. The software is particularly focussed on so-called *frequent* services, which means that more than six buses are scheduled to depart per hour. In this case, passengers are not expected to base their decision of when to arrive at a stop on an explicit timetable. Instead, performance is expressed in terms of the regularity of the *headways*, i.e., the inter-departure times at stops.

The parametric model allows for the analysis of the impact on headway regularity of different strategies, such as real-time headway control. The headway regularity measures are studied in steady state. One example is the Buses-Per-Hour (BPH) metric, which at any time point t is 0 if six or more buses arrived at a given location (e.g., a bus stop or the end of a journey stage) in the previous hour and 1 otherwise. Scottish government regulations stipulate that the steady-state BPH value should be at most 5%; meaning that a regulator arriving at the given location in steady state should observe six bus departures in the next hour with a probability of over 95%. The counting behaviour of the regulator is represented in the simulation by a forgetful observer automaton which behaves as described in Figure 4.

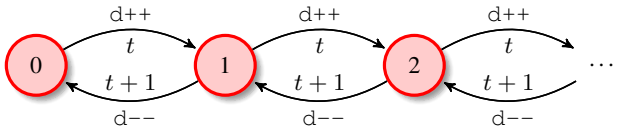

Fig. 4. Observers counting departures should note the times of departures and forget departures which are more than one hour old.

The BPH metric is expressed within MultiQuaTEx as displayed in Listing 4, where H_8 is a state observation evaluated within the bus simulator as the number of buses which arrived in the previous hour at the end of stage 8.

```
1 BPH8() = if {s.rval("H_8") < 6}
2                 then 1
3                 else 0
4          fi;
5
6 eval batchMeans(E[ BPH8() ]);
```

Listing 4. The MultiQuaTEx query for the BPH after stage/patch 8.

One question that is of interest to planners is how many buses need to be assigned to the route to meet the government regulations. An overview of the performance of a specific bus service in Edinburgh (namely the Airlink service) for different numbers of buses assigned to the route can be found in Table 1.

Each row of Table 1 contains estimates of the BPH metric at the end of a specific stage of the route roughly corresponding to the end of patch 8 in Figure 5. This location is immediately after a busy junction in the city centre and the value of the BPH metric here is higher than at most other points along the route. The BPH metric expresses a penalty and so a higher value of the BPH metric signifies worse performance. The requirement for the frequent services such as the Airlink is that the width of the confidence interval around the satisfaction of the BPH requirement should be at most 0.05 — we see from the table that at least 9 buses are required to meet this criterion.

#buses	C.I.	# iterations	runtime (s)
8	[0.5095, 0.5100]	4	306.2
9	[0.0110, 0.0119]	3	193.5

Table 1. Analysis of the performance of the Airlink service in Edinburgh for different numbers of buses assigned to the route.

We display the BPH at the end of a specific stage — corresponding to a so-called *patch* — of the route. An example of a patch structure for the Airlink route is displayed in Figure 5.

Fig. 5. One possible stage structure along the Airlink route in Edinburgh.

The default number of buses servicing the Airlink route is 11, and the value of the BPH metric is very low in this setting because, with so many buses serving the route, it is very unlikely that there will be fewer than six buses arriving in any given hour. From the table we see that government regulations would still be met if two buses were removed from the route: 11 buses are used, but 9 would be sufficient. Our analysis thus suggests that the bus company which operates this service could cut fuel costs and reduce atmospheric emissions of pollutants while continuing to satisfy the relevant performance requirements imposed in legislation.

6 Conclusions

In this paper we have extended a statistical model-checker with computational methods for the calculation of steady-state properties of dynamic systems viewed as black-box systems which offer no access to their internal structure or logic. The challenges of this setting are that we have only the perspective of an external observer, and can only express properties over the model outputs. The benefits of this approach are that the method is applicable to a wide range of simulators, across a wide range of modelling formalisms. In this paper we computed steady-state metrics of a model in a biochemical reaction simulator and a transportation system in a custom discrete-event simulator.

Working across formalisms in this way means that the MultiVeStA tool provides a transferrable statistical model-checking service which modellers can apply to their own preferred modelling formalism, allowing them to continue working with the modelling languages where they have accumulated experience, or which are best suited to the problem which is under study. In the future, we plan to further enrich the family of analysis techniques offered by MultiVeStA, and to apply the tool to further domains.

The provision of a query language such as MultiQuaTEx also supports the transferability of the MultiVeStA service to other formalisms in a way in which a specialised logic would not. Based on functions, conditional statements, and arithmetic expressions, MultiQuaTEx resembles a programming language more than a logic, making

understanding and expressing properties easier for a practitioner because of the more accessible syntax which is used. In the future we plan to develop an editor for Multi-QuaTEx, offering also the possibility of selecting among a set of predefined queries.

Acknowledgments This work has been supported by the EU project QUANTICOL, 600708. The authors thank Bill Johnston and Philip Lock of Lothian Buses for providing access to the data and for their helpful feedback on parts of this research project, and Jane Hillston for her helpful comments.

References

1. G. A. Agha, J. Meseguer, and K. Sen. PMaude: Rewrite-based specification language for probabilistic object systems. In *QAPL 2005*, volume 153(2) of *ENTCS*, pages 213–239. Elsevier, 2006.
2. C. Alexopoulos and D. Goldsman. To batch or not to batch? *ACM Transactions on Modeling and Computer Simulation*, 14(1):76–114, 2004.
3. C. Alexopoulos and A. F. Seila. Implementing the batch means method in simulation experiments. In *Proceedings of the 28th Conference on Winter Simulation*, WSC '96, pages 214–221, Washington, DC, USA, 1996. IEEE Computer Society.
4. M. AlTurki and J. Meseguer. PVeStA: A parallel statistical model checking and quantitative analysis tool. In A. Corradini, B. Klin, and C. Cîrstea, editors, *CALCO 2011*, volume 6859 of *Lecture Notes in Computer Science*, pages 386–392. Springer, 2011.
5. S. Arora, A. Rathor, and M. V. P. Rao. Statistical model checking of opportunistic network protocols. In *Proceedings of AINTEC 2015*, pages 62–68. ACM, 2015.
6. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model-checking continuous-time Markov chains. *ACM Transactions on Computational Logic (TOCL)*, 1(1):162–170, 2000.
7. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE TSE*, 29(6):524–541, 2003.
8. M. Bartoletti, T. Cimoli, M. Murgia, A. S. Podda, and L. Pompianu. A contract-oriented middleware. In *FACS 2015*, pages 86–104, 2015.
9. L. Belzner, R. De Nicola, A. Vandin, and M. Wirsing. Reasoning (on) Service Component Ensembles in Rewriting Logic. In *Specification, Algebra, and Software*, volume 8373 of *LNCS*, pages 188–211. Springer, 2014.
10. L. Belzner, R. Hennicker, and M. Wirsing. Onplan: A framework for simulation-based online planning. In *FACS 2015, Revised Selected Papers*, pages 1–30, 2015.
11. B. Boyer, K. Corre, A. Legay, and S. Sedwards. Plasma-lab: A flexible, distributable statistical model checking library. In *QEST 2013*, pages 160–164. Springer, 2013.
12. A. Bušić, B. Gaujal, and J.-M. Vincent. Perfect simulation and non-monotone Markovian systems. In *Valuetools 2008*. ICST, 2008.
13. V. Ciancia, D. Latella, M. Massink, R. Paskauskas, and A. Vandin. A tool-chain for statistical spatio-temporal model checking of bike sharing systems. In *ISoLA 2016*, volume 9952 of *LNCS*, pages 657–673, 2016.
14. F. Ciocchetta, A. Duguid, S. Gilmore, M. L. Guerriero, and J. Hillston. The bio-pepa tool suite. In *QEST 2009*, pages 309–310, Sept 2009.
15. F. Ciocchetta, A. Duguid, and M. L. Guerriero. A compartmental model of the cAMP/P-KA/MAPK pathway in Bio-PEPA. In G. Ciobanu, editor, *MeCBIC 2009*, volume 11 of *EPTCS*, pages 71–90, 2009.
16. F. Ciocchetta and J. Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *TCS*, 410(33-34):3065–3084, 2009.

17. R. W. Conway. Some tactical problems in digital simulation. *Management Science*, 10(1):47–61, 1963.
18. D. El Rabih and N. Pekergin. Statistical model checking using perfect simulation. In *ATVA 2009*, pages 120–134. Springer, 2009.
19. N. Gast, G. Massonnet, D. Reijsbergen, and M. Tribastone. Probabilistic forecasts of bike-sharing systems for journey planning. In *CIKM 2015*, pages 703–712, 2015.
20. S. Gilmore, M. Tribastone, and A. Vandin. An Analysis Pathway for the Quantitative Evaluation of Public Transport Systems. In *IFM 2014*, pages 71–86, 2014.
21. P. W. Glynn and W. Whitt. The asymptotic validity of sequential stopping rules for stochastic simulations. *The Annals of Applied Probability*, pages 180–198, 1992.
22. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
23. J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance evaluation*, 68(2):90–104, 2011.
24. P. L'Ecuyer. SSJ: Stochastic simulation in Java, software library, 2016. `http://simul.iro.umontreal.ca/ssj/`.
25. P. L'Ecuyer, L. Meliani, and J. Vaucher. SSJ: A framework for stochastic simulation in Java. In E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, editors, *Proceedings of the 2002 Winter Simulation Conference*, pages 234–242. IEEE Press, 2002.
26. A. Legay, B. Delahaye, and S. Bensalem. Statistical model checking: An overview. In *RV 2010*, pages 122–135. Springer, 2010.
27. S. R. Neves, P. Tsokas, A. Sarkar, E. A. Grace, P. Rangamani, S. M. Taubenfeld, C. M. Alberini, J. C. Schaff, R. D. Blitzer, I. I. Moraru, and R. Iyengar. Cell Shape and Negative Links in Regulatory Motifs Together Control Spatial Information Flow in Signaling Networks. *Cell*, 133(4):666 – 680, 2008.
28. D. Pianini, S. Sebastio, and A. Vandin. Distributed statistical analysis of complex systems modeled through a chemical metaphor. In *HPCS 2014*, pages 416–423. IEEE, 2014.
29. D. Reijsbergen and S. Gilmore. An automated methodology for analysing urban transportation systems using model checking. *https://danielreijsbergen.files.wordpress.com/2016/10/bus_modelling1.pdf*, 2016.
30. S. Sebastio, M. Amoretti, and A. Lluch Lafuente. A Computational Field Framework for Collaborative Task Execution in Volunteer Clouds. In *SEAMS 2014*, pages 105–114. ACM, 2014.
31. S. Sebastio and A. Vandin. MultiVeStA: Statistical model checking for discrete event simulators. In *Valuetools 2013*, pages 310–315. ACM, 2013.
32. K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of black-box probabilistic systems. In *CAV 2004*, pages 202–215. Springer, 2004.
33. K. Sen, M. Viswanathan, and G. A. Agha. Vesta: A statistical model-checker and analyzer for probabilistic systems. In *QEST 2005*, pages 251–252, 2005.
34. N. M. Steiger, E. K. Lada, J. R. Wilson, J. A. Joines, C. Alexopoulos, and D. Goldsman. ASAP3: A batch means procedure for steady-state simulation analysis. *ACM Transactions on Modeling and Computer Simulation*, 15(1):39–73, 2005.
35. M. H. ter Beek, A. Legay, A. Lluch-Lafuente, and A. Vandin. Statistical analysis of probabilistic models of software product lines with quantitative constraints. In *SPLC 2015*, pages 11–15. ACM, 2015.
36. M. H. ter Beek, A. Legay, A. Lluch-Lafuente, and A. Vandin. Statistical model checking for product lines. In *ISoLA 2016*, volume 9952 of *LNCS*, pages 114–133, 2016.
37. H. L. Younes. Probabilistic verification for black-box systems. In *CAV 2015*, pages 253–265. Springer, 2005.
38. H. L. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV 2002*, pages 223–235. Springer, 2002.